PostgreSQL Architecture

2024. 06. 10　PostgreSQL

PostgreSQL Architecture

PostgreSQL 의 Architecture 구조는 매우 단순한 편이다. 공유 메모리, 적은 수의 백 그라운드 프로세스와 데이터 파일로 구성된다. (그림 1-1. 참조)

그림 1-1. PostgreSQL Architecture

1-1. Shared Memory

Shared Memory에서 가장 중요한 요소는 Shared Buffer와 WAL Buffer이다.
Shared Buffer
Shared Buffer의 목적은 모든 데이터베이스가 그렇듯 DISK I/O 를 최소화하는 것이다. 이를 위해, 아래 항목을 만족해야 한다.

- 매우 큰(수십, 수백 GB) 버퍼를 빠르게 액세스해야 한다.
- 많은 사용자가 동시에 접근할 때 경합을 최소화 해야 한다.
- 자주 사용되는 블록은 최대한 오랫동안 버퍼 내에 있어야 한다.

WAL Buffer
WAL Buffer는 데이터베이스의 변경 사항을 잠시 저장하는 버퍼이다. WAL Buffer내에 저장된 내용은 정해진 시점에 WAL 파일로 기록된다. 백업 및 복구 관점에서 보면, WAL Buffer와 WAL 파일은 매우 중요하다. (Oracle의 Redo Log Buffer와 Redo Log File 처럼)

1-2. Process 유형

PostgreSQL의 프로세스 유형은 4 가지이다.

- Postmater (Daemon) 프로세스
- Background Process
- Backend Process
- Client Process

1-3. Postmaster Process

Postmaster Process는 PostgreSQL을 기동할 때 가장 먼저 시작되는 프로세스이다. 초기 기동 시에 복구 작업, Shared 메모리 초기화 작업, 백 그라운드 프로세스 구동 작업을 수행한다. 또한, 클라이언트 프로세스의 접속 요청이 있을 때 Backend Process를 생성한다.

그림 1-2. Process 관계도

pstree 명령어로 프로세스 간의 관계를 확인하면, Postmaster 프로세스가 모든 프로세스의 부모 프로세스인 것을 확인할 수 있다. (이해를 돕기 위해, 프로세스 ID 뒤에 프로세스 명과 Argument 를 추가 했다.)

```
[postgres@localhost pg11]$ pstree -p 20936
postgres(20936) /home/postgres/pg11/bin/postgres -D /data/pg11
┬──postgres(20937) postgres: logger
├──postgres(20939) postgres: checkpointer
├──postgres(20940) postgres: background writer
├──postgres(20941) postgres: walwriter
```

```
├─postgres(20942) postgres: autovacuum launcher
├─postgres(20943) postgres: stats collector
└─postgres(20944) postgres: logical replication launcher
```

1-4. Backgroud Process

PostgreSQL 운영에 필요한 백그라운드 프로세스 목록은 다음과 같다. Autovacuum laucher를 제외한 나머지 프로세스들은 ORACLE에서도 쉽게 찾아볼 수 있는 것들이다.

Process 명	수행하는 일
logger	에러 메세지 및 다양한 정보를 로그 파일에 기록 한다.
checkpointer	체크포인트 발생 시, dirty 버퍼를 파일에 기록한다.
wirter	주기적으로 dirty 버퍼를 파일에 기록한다.
wal writer	WAL Buffer 내용을 WAL 파일에 기록한다.
autovacuum launcher	Vacuum이 필요한 시점에 autovacuum worker를 fork한다.
archiver	Achive log 모드일 때, WAL 파일을 지정된 디렉토리에 복사한다.
stats collector	세션 수행 정보 (pg_stat_activity)와 테이블 사용 통계 정보 (pg_stat_all_tables) 와 같은 DBMS 사용 통계 정보를 수집한다.

1-5. Backend Process

Backend Process의 최대 개수는 max_connections 파라미터로 설정하며, 기본 설정 값은 100이다.
Backend Process는 사용자 프로세스 쿼리 요청을 수행한 후, 결과를 전송하는 역할을 수행한다. 쿼리 수행에 필요한 몇 가지 메모리 구조가 필요한데 이것을 통칭해서 로컬메모리라고 한다. 로컬메모리와 관련된 주요 파라미터는 다음과 같다.
work_mem 파라미터
정렬 작업, Bitmap 작업, Hash조인과 Merge 조인 작업 시에 사용되는 공간이다. 기본 설정값은 4MB이다.
maintenance_work_mem 파라미터
Vacuum 및 Create Index 작업 시에 사용되는 공간이다. 기본 설정값은 64MB이다.
temp_buffers 파라미터
Temporary 테이블을 저장하기 위한 공간이다. 기본 설정값은 8MB이다.

1-6. Database 구조

Database 관련 사항

1. PostgreSQL은 여러 개의 데이터베이스로 구성된다. 이를 데이터베이스 클러스터라고 한다.
2. intidb() 수행 시에 template0, template1, postgres 데이터베이스가 생성된다.
3. template0과 template1 데이터베이스는 사용자 데이터베이스 생성을 위한 템플릿 데이터베이스이며 시스템 카탈로그 테이블을 포함하고 있다.
4. initdb() 수행 직후에 template0 과 template1 데이터베이스의 테이블 목록은 같다. 단 template1의 데이터베이스에는 사용자가 필요한 오브젝트를 생성할 수 있다.
5. 사용자 데이터베이스를 생성할 때 template1 데이터베이스를 복제한다. 이점을 이용하면, 사용자 데이터베이스마다 특정한 오브젝트를 매번 생성하는 번거로움을 해결할 수 있다.

Tablespace 관련 사항

1. initdb() 수행 직후에 pg_default, pg_global Tablespace 가 생성된다.
2. 테이블 생성 시에 Tablespace를 지정하지 않으면 pg_default Tablespace 에 저장된다.

3. 데이터베이스 클러스터 레벨에서 관리되는 테이블은 pg_global Tablespace 에 저장된다.
4. pg_default Tablespace의 물리적 위치는 $PGDATA\base 이다.
5. pg_global Tablespace의 물리적 위치는 $PGDATA\global 이다.
6. 1개의 Tablespace를 여러 개의 데이터베이스가 사용할 수 있다. 이때, Tablespace 디렉토리 내에 Database별 서브 디렉토리가 생성된다. 이때 서브 디렉토리의 이름은 Database의 OID가 된다.
7. 사용자 Tablespace를 생성하면 $PGDATA\tblspc 디렉토리에 사용자 Tablespace와 관련된 심볼릭 링크가 생성된다.

Table 관련 사항

1. 테이블 별로 3개의 파일이 존재한다.
2. 1개는 테이블 데이터를 저장하기 위한 파일이다. 파일명은 테이블의 OID이다.
3. 1개는 테이블 여유 공간을 관리하기 위한 파일이다. 파일명은 OID_fms 이다.
4. 1개는 테이블 블록의 visibility를 관리하기 위한 파일이다. 파일명은 OID_vm 이다.
5. 인덱스는 vm 파일이 없다. 즉, OID, OID_fsm 2개의 파일로 구성된다.

Note. 테이블과 인덱스 생성 시점의 파일 명은 OID 이며 이 시점에 OID와 pg_class.relfilenode의 값은 같다. 하지만 Rewrite 작업 (Truncate, Cluster, Vacuum Full, Reindex 등)이 수행되면, 영향 받는 오브젝트의 relfilenode 값이 변경 되고, 파일 명 또한 relfilenode 값으로 변경된다. 참고로, pg_relation_filepath('object명') 함수를 이용하면 파일 위치와 이름을 쉽게 확인할 수 있다.
tempate0, template1, postgres 데이터베이스
앞서 살펴본 내용을 테스트를 통해 확인해보자.
initdb() 수행 후에 pg_database 뷰를 조회하면, template0, tempate1, postgres 데이터베이스가 생성된 것을 확인할 수 있다.

```
postgres=# select oid, datname, datistemplate, datallowconn from pg_database order by 1;
oid  |  datname  | datistemplate | datallowconn
—-+——+———+———
1 | template1 | t            | t
13286 | template0 | t            | f
13287 | postgres  | f            | t
(3 rows)
```

- datistemplate 컬럼을 통해서 template0과 template1 데이터베이스는 사용자 데이터베이스 생성을 위한 template용 데이터베이스라는 사실을 알 수 있다.
- datallowconn 컬럼은 데이터베이스 접속 가능 여부를 알려준다. template0 데이터베이스는 접속할 수 없으므로, 해당 데이터베이스의 내용 또한 변경할 수 없다.
- template용 데이터베이스를 2개 제공하는 이유는 template0 데이터베이스는 초기 상태 템플릿을, template1 데이터베이스는 사용자가 추가한 템플릿을 제공하기 위함이다.
- postgres 데이터베이스는 template1 데이터베이스를 이용해서 생성된 기본 데이터베이스이다. 접속 시에 데이터베이스를 지정하지 않으면 postgres 데이터베이스로 접속된다.

데이터베이스는 $PGDATA/base 디렉토리 아래에 위치한다. 디렉토리 명은 데이터베이스 OID 번호이다.

```
[postgres@localhost base]$ ls -l
total 36
drwx——. 2 postgres postgres 8192 Dec 11 06:04 1
drwx——. 2 postgres postgres 8192 Dec 11 06:04 13286
drwx——. 2 postgres postgres 8192 Dec 11 06:06 13287
```

사용자 데이터베이스 생성

사용자 데이터베이스는 template1 데이터베이스를 복제해서 생성된다. 이를 확인하기 위해서 template1 데이터베이스에 사용사 테이블 T1을 생성한다. 그리고 mydb01 데이터베이스를 생성한 후에, T1 테이블이 존재하는지 확인해 보자.

```
– template 데이터베이스에 접속해서 T1 테이블을 생성한다.
template1=# create table t1 (c1 int);
CREATE TABLE
```

– mydb01 데이터베이스를 생성한다.
postgres=# create database mydb01;
CREATE DATABASE
– mydb01 데이터베이스에 접속한 후 T1 테이블이 존재하는지 확인한다
mydb01=# \\dt
List of relations
Schema | Name | Type | Owner
——+—–+—–-+——-
public | t1 | table | postgres
(1 row)

이를 그림으로 표현하면 다음과 같다.

그림 1-3 템플릿 데이터베이스와 사용자 데이베이스 간의 관계

pg_default 테이블스페이스
initdb() 수행 후에 pg_tablespace를 조회하면 pg_default와 pg_global 테이블스페이스가 생성된 것을 확인할 수 있다.

```
postgres=# select oid, * from pg_tablespace;
oid  |  spcname   | spcowner | spcacl | spcoptions
—–+———-+——-+—–+———
1663 | pg_default |       10 |        |
1664 | pg_global  |       10 |        |
(2 rows)
```

- pg_default 테이블스페이스의 위치는 $PGDATA\base이다. 해당 디렉토리에는 데이터베이스 OID 별 서브 디렉토리가 존재한다.
- 즉, 물리적 구성으로 보면 테이블스페이스 하위에 해당 테이블스페이스를 사용하는 데이터베이스 디렉토리가 존재하는 것이다.(그림 1-4 참조)

```
[postgres@localhost base]$ ls -l
total 48
drwx——. 2 postgres postgres 8192 Dec 11 06:17 1
drwx——. 2 postgres postgres 8192 Dec 11 06:04 13286
drwx——. 2 postgres postgres 8192 Dec 11 06:06 13287
drwx——. 2 postgres postgres 8192 Dec 11 06:19 16387– 데이터베이스들의 OID 확인
postgres=# select oid, datname from pg_database order by 1;
oid  |  datname
——-+———
1 | template1
13286 | template0
13287 | postgres
16387 | mydb01
(4 rows)
```

그림 1-4 물리적 구성 관점에서 본 pg_default 테이블스페이스와 데이터베이스 관계

pg_global 테이블스페이스

pg_global 테이블스페이스는 '데이터베이스 클러스터' 레벨에서 관리 해야 할 데이터들을 저장하는 용도의 테이블스페이스이다.

- 예를 들어, pg_database 테이블과 같은 유형의 테이블들은 어떤 데이터베이스에서 접속해서 조회하더라도 동일한 정보를 제공한다. (그림 1-5. 참조)
- pg_global 테이블스페이스의 위치는 $PGDATA\global 이다.

그림 1-5 pg_global 테이블스페이스와 데이터베이스간의 관계

사용자 테이블스페이스 생성
사용자 테이블스페이스를 생성한 후 변경사항을 확인해보자.

```
– user 생성
create role hr password hr;
```

– 물리 서버에 directory 생성
sudo mkdir -p ./tablespaces/hr
sudo chown -R postgres ./tablespaces/hr
– owner를 hr 두고 tablespace 생성
create tablespace hr owner hr location '/data/tablespaces/hr';
– 해당 tablespace에 table 생성
create table test (col1 text, col2 text) tablespace hr;

PostgreSQL에서 DB는 PGDATA라는 환경 변수에 지정된 디렉토리를 통째로 DB로 사용한다. 그리고 그 하위에 테이블이 파일로 생성된다. 즉 테이블스페이스를 따로 생성하지 않아도 DB에 사용자 테이블을 만들 수 있다. 기본적으로 DB로 지정된 디렉토리 전체가 하나의 기본 테이블스페이스로 인식되는 것이다.

PostgreSQL 공식사이트에서는 다음과 같이 테이블스페이스를 정의하고 있다.
PostgreSQL 에서 테이블스페이스는 DB관리자에 의해 데이터베이스의 객체가 저장될 수 있는 파일 시스템의 경로로 정의된다. 테이블스페이스가 생성되면 데이터베이스 객체에 객체를 생성할 때 이름에 의해서 테이블스페이스가 참조될 수 있다.
PostgreSQL이 설치될 때 디스크 레이아웃에 따라 관리자가 생성할 수 있는데 두 가지 관점에서 매우 유용하다.

- 첫번째는 DB가 생성된 볼륨 또는 파티션에 여유 공간이 부족할 때 테이블스페이스를 다른 파티션이나 디스크에 생성하여 시스템을 재구성할 때까지 DB를 확장할 수 있다.
- 두번째는 데이터베이스 객체의 성능 최적화를 위해 사용할 수 있다는 것이다. 예를 들어 매우 사용량이 많으면서 자주 업데이트 되는 인덱스를 위해 고성능 SSD를 장착하고테이블스페이스를 생성하여 인덱스를 SSD에 생성하여 성능을 개선하는데 사용될 수 있다.